Software Acquisition

TASK DELIVERABLE:

Final Report

Submitted By: Charles Coleman
Manager, SATC

December 2003

Technical POC: Al Gallo
Phone #: 301-286-3756
Fax #: 301-286-1667
Email: al.gallo@gsfc.nasa.gov
Mail Code: 304

Administrative POC: Dennis Brennan
Phone #: 301-286-6582
Fax #: 301-286-1667
Email: Dennis.Brennan.1@gsfc.nasa.gov
Mail Code: 300.0

Table of Contents

1 The problem

Obtaining useful software is a long-standing problem in both the public and private sectors. Despite the considerable attention and resources directed toward this problem, the fact remains that only a small percentage of software development projects produce software that is put into productive use.

Three factors combine to exacerbate this problem within the NASA community. The first factor is the ever-increasing trend to acquire software via contract rather than develop that software in-house. The second factor is the move towards *performance based contracts* and other acquisition vehicles that focus the Agency's attention on the requirements for the software product being acquired while leaving it up to the contractor to determine how those requirements are to be met. The third factor is the increasing use of software to provide critical system functions.

1.1 Acquiring software via contract rather than develop software in-house

Acquiring software from an external source places several barriers between those that need the software and those that build the software. Each barrier (e.g., organizational, cultural, technical, economic, distance) increases the risks that the software delivered will not meet the needs and expectations of prospective users.

Risk management (risk identification, assessment, and mitigation) is an important method for assuring that contractually acquired software will meet delivery, performance, and quality requirements. This risk – based approach requires the availability of critical items of information to objectively determine the status of deliverable products and to identify trends that may impact the availability and utility of those products. *What is needed is the identification of the information critical for managing the technical risks associated with acquiring software from an external source*

Identifying and getting relevant data is just part of the solution to managing the risks associated with contractually acquired software. The real challenge is in using the data to objectively gain insight on probable risks and provide the appropriate level of oversight for the software acquisition. At any given time, the contractor may provide a large number of information items to the acquirer. Each information item provides a measure relevant for assessing some aspect of software development risk. Looking at each information item in isolation is not likely to be useful. *What is needed is some way of analyzing the data and presenting the results of that analysis in a form that clearly communicates the areas and degree of the risks.*

1.2 Performance-Based Contracts

One of the key aspects of performance-based contracts involves strict separation of what work is to be done from how that work is to be performed. The acquirer specifies the requirements (what) while leaving it up to the contractor to determine how those requirements are to be met.

Relying on requirement specifications as the sole means of communicating the salient attributes desired of a software product is as much a problem today as it has always been. Although developing high quality requirement specifications is not sufficient to insure software quality, developing high quality requirement specifications is an important first step towards achieving software quality.

Current methods and tools for developing high quality requirement specifications typically requires as much effort by the acquirer to specify what is to be done, as it does for the developer to actually do the work. *What is needed is a method, supported by automated tools, that achieves a better balance between developing the requirements and implementing the requirements.*

1.3 Use of software to provide critical system functions

The increase in technical complexity of NASA missions can be attributed, in large part to, to the increasing use of software to provide critical system functions. NASA projects have ultimate responsibility for the safety and assurance of the mission in spite of the fact that a contact and contractors may be involved. Assuring that software, which provides critical system functions, performs as required is challenging in its own right. The fact that a contract and contractors are involved significantly complicates the task. *What is needed is a method to improve the process by which NASA acquires critical system software.*

2 Research Objective

The problems associated with the three factors described above are not going away. In fact, budget and other pressures are likely to make those problems more difficult to deal with in the future than they are today.

The objective of the research is to identify methods and tools that NASA projects can use to mitigate some of the adverse effects of the problem of acquiring software from external sources. Underlying this objective is the hypothesis that there are viable

methods/tools that can enable NASA project managers to mitigate a significant number of the major risks associated with acquiring software from external sources.

It is important to note that our focus is on ***acquiring software via contract*** as opposed to ***acquiring software from contractors***. In many NASA environments, support contactors are indistinguishable from NASA staff. In these environments, there is typically not a contract-imposed risk beyond those risks usually associated with a software development activity.

3 Research Approach

We set out to explore three broad classes of methods/tools.

The first class focused on methods/tools that provide NASA projects with improved insight and oversight into the contractor's activities. Key to gaining and effectively using this insight/oversight is identifying the essential items of information that NASA projects should obtain from the contractor (e.g., problem reports). Rather than just identifying a laundry list of data items, we wanted to establish a framework that provides a logical structure and context for the data items. *Much of our effort went into defining this structural and contextual framework.*

The second class dealt with methods/tools that help NASA projects improve the quality of requirement specifications. Our original thoughts were to explore the use of UML[1] to increase the clarity and precision of requirement specifications, and to find important and subtle errors. During the course of our research it became evident that requirement clarity and precision was a much broader problem than just the use of a particular modeling language. *After identifying a technology that shows promise as an approach to the broader problem, we did not further pursue this element of the research.*

The third class involved methods/tools that help projects objectively assess, and thus improve, their software acquisition capability. Of particular interest and potential applicability is the Software Acquisition Capability Maturity Model (SA-CMM)[2]. *Much*

---

[1] The Unified Modeling Language™ (UML); http://www.rational.com/uml/index.jsp?SMSESSION=NO

[2] Jack Ferguson, et all, "Software Acquisition Capability Maturity Model (SA-CMM) Version 1.01"; Technical Report CMU/SEI-96-TR-020; December 1996

of our effort went into identifying critical factors underlying the successful use of SA-CMM.

4 Research Results

4.1 Essential Items of Information Framework

Our research identified six common areas of software development risks. Each of these risk areas is characterized by one or more factors associated with the development processes and/or software products. The factors that characterize each risk area are defined by one or more measures. This relationship between risks, factors contributing to those risks, and measures for assessing those factors form the framework for identifying the essential items of information needed to objectively assess and manage risks associated with software development. This framework is similar to the Goal, Question, Metric[3] paradigm developed by Basili and Weiss.

Two points regarding our criteria for selecting the areas of software development risks. First of all, the areas were deemed to be relevant to the NASA experience. Secondly, the areas relate specifically to acquiring software from an external source

The six common areas of software development risks are:

1. Schedule/Progress
2. Development Resources
3. Product Growth and Stability
4. Product Quality
5. Development Performance
6. Technical Adequacy

Measures for each factor of each risk area are identified and quantified by specific items of information or data acquired from sources that must be included in the contract's list of deliverables. The table below illustrates the use of the framework. Here, an area of risk (Schedule/Progress) is shown with one of the factors (Milestone Performance) associated with that area of risk along with the information used to assess that factor.

---

[3] Basili, V. R. and Weiss D. M. "*A Methodology for Collecting Valid Software Engineering Data*." IEEE Transactions on Software Engineering, 10(6), 1984, 728-738.

Risk Area	Risk Factor	Risk Measures
Schedule/Progress *{Progress against an established development and delivery schedule}*	Milestone Performance *{Monitoring changes to the milestone schedule enables the project manager to assess the potential risk that scheduled future project milestone may not be achieved}*	Names Of Planned Activities & Events Dependencies Between Activities/Events Scheduled Milestone Event Dates Dates That Milestone Events Actually Occur Number Of Times Each Event Has Been Rescheduled 1. Name/ID# Of Design Components 2. Scheduled Start Date Of Each Component 3. Actual Start Date Of Each Component 4. Scheduled Completion Date Of Each Component 5. Actual Completion Date Of Each Component

The complete list of risk areas, associated factors, and information items are presented in Task Deliverable: *Essential Items Of Information That NASA Projects Should Obtain From The Contractor* March 31, 2003.

In conjunction with the essential items of information framework, we also developed a framework for

- identifying the functionality of a toolset for analyzing the essential items of information, and
- presenting the results of that analysis in a form that clearly communicates the areas and degree of the risks.

The requisite functionality involves a multivariate analysis of the risk measures to arrive at an overall assessment of the software development risk. A description of the multivariate analysis is presented in Task Deliverable: *Analysis of the Essential Items of Information That NASA Projects Should Obtain From The Contractor* June 30, 2003.

Regarding the two frameworks; it is useful to note that;

- for a given environment, there may be other areas of software development risk that are deemed more important that the ones identified here. The value of the essential items of information framework is that it can be adapted to any set of software development risks
- there is nothing magical about six areas of risks. The framework for identifying the functionality of a toolset for assessing the essential items of information can be adapted to fit the particular number of risk areas relevant for a given environment.

4.2 Methods/Tools for Requirement Clarity and Precision

As stated earlier, our original thoughts were to explore the use of UML to increase the clarity and precision of requirement specifications, and to find important and subtle errors. During the course of our research it became evident that requirements clarity and precision was a much broader problem than just the use of a particular modeling language. In trying to understand the broader problem we identified two key issues:

- modeling,
- methods/tools for deriving useful information from the models.

In our attempts to more precisely relate these issues to the problem of requirements clarity and precision, we came across a technology known as 001[4]. 001 has a large number of interesting features applicable to the all phases of software development lifecycle. We, however, limited our focus on those features that

- relate directly to issues of requirements clarity and precision
- are important in an environment where the software is acquired from an external source.

4.2.1 Overview

001 is a collection of concepts, methods, and tools ostensibly developed to automate the paradigm, Development Before the Fact (DBTF). There is a language, 001 AXES, which supports the representation of DBTF and a set of tools, the 001 Tool Suite, which supports the application and use of DBTF.

The chief claim of DBTF is that it is a ***preventive paradigm***, that is, problems associated with traditional methods of design and development are prevented "before the fact" just by the way a system is defined. This is contrast to the traditional ***curative paradigm*** that focuses on finding and fixing problems after they've surfaced -- often at the most inopportune and expensive point in time.

The benefits of 001 derive primarily from the models that are created using the 001 AXES language. After a model is defined, the 001 Tool Suite can be used to analyze the modeled system and to automate functions such as code generation and documentation. Although not yet implemented, the 001 team has designed a reverse engineering tool that can be used to document, understand, and eventually bring legacy code under the 001 umbrella.

4.2.2 Modeling

Modeling boils down to the questions of fidelity, scalability and the underlying systems theory upon which the modeling concepts are based.

---

[4] Margaret H. Hamilton and William R. Hackler, “Towards Cost Effective and Timely End-to End Testing”, prepared for Army Research Laboratory, Georgia Tech.; Contract No. DAKF11-99-P-1236; July 17, 2000

With respect to fidelity, 001 models are built using ***system-oriented objects*** (SOOs). SOOs are imbued with all of the detail that allows it to be understood without ambiguity by all other objects within the model. 001 comes with a standard set of low-level SOOs. Higher-level SOOs can be constructed from lower-level SOOs.

Questions regarding scalability become issues of complexity. Small demo systems are clearly easier to model than are large heterogeneous, geographically distributed, real-time, system. It's not clear how the 001 AXES language assists in dealing with issues of complexity. In addition, we are not aware of an application in which the 001 AXES language has been used to model large heterogeneous, geographically distributed, real-time, systems.

The underlying systems theory upon which the modeling concepts are based appear to have evolved in *whole cloth* from the extensive experience of Margaret Hamilton in building and analyzing systems. UML, by way of contrast, has its roots in the internationally developed Open Distributed Processing[5] systems model. The 001 team, however, is intimately familiar with UML and has done extensive comparisons highlighting similarities and differences. 001 did not suffer in those comparisons.

4.2.3 Tools

Tools boil down to the scope of functionality embodied in the tool, ease of use, and the degree of coupling between the tool functionality and the underlying systems theory.

The 001 tool suite appears to cover the full spectrum of life cycle functions, including: requirements and design modeling; automatic code generation; test and execution; and simulation. In addition, 001 can be used to coexist and interface with other tools

With respect to ease of use, the 001 team claims that the tool suite is no more difficult to use than other comprehensive tool suites. In fact, they have documented comparisons that suggest that the 001 tool suite functions are easier to use than corresponding functions in systems such as UML. Its one thing to measure ease of use within a

---

[5] Jan de Meer, Bernd Mahr, Silke Storp (Eds.): Open Distributed Processing, II: Proceedings of the IFIP TC6/WG6.1 International Conference on Open Distributed Processing, Berlin, Germany, 13-16 September 1993. IFIP Transactions C-20 North-Holland 1994, ISBN 0-444-81861-8

population of users comfortable with abstractions and modeling – its quite another to measure ease of use within a population of system development practitioners.

There is a very tight coupling between the 001 tool suite functionality and systems theory underlying DBTF. For example Fmaps and Tmaps are key systems theory artifacts that are tightly coupled to 001 tool suite functionality. This tight coupling is a two-edged sword – on the one hand it facilitates use of the tool suite by users steeped in the underlying systems theory; on the other hand it can be a significant barrier to the use of the tool suite by users unfamiliar with the underlying systems theory.

4.3 SA-CMM Critical Success Factors

The experience of the Software Engineering Institute in developing the Capability Maturity Model for Software (SW-CMM) was directly applicable to developing the SA-CMM. The SW-CMM describes the contractor's role while the SA-CMM describes the acquirer's role in the software acquisition process. The SA-CMM identifies key process areas for four of its five levels of maturity. The key process areas state the goals that must be satisfied to achieve each level of maturity. The SA-CMM is designed to be generic enough for use by any government or industry organization, regardless of size, acquiring software. When applying the SA-CMM to a particular organization, translations may need to be made in addition to tailoring the model to fit a specific acquisition.

Our research identified the following factors that are critical to the successful application of SA-CMM.

- Visible, high-level management commitment to improve the acquisition of software intensive systems
- A centrally coordinated/supported acquisition function to serve as the repository of SA-CMM experience and expertise
- Access to SEI expertise to assist in training and transition to the use of SA-CMM
- Sufficient staffing and funding for a well-defined, well coordinated SA-CMM effort (no real rule of thumb yet established)
- A convincing ROI value proposition and/or recognition that the organization has a problem that can be addressed by SA-CMM

Further, we found that the SA-CMM can be viewed from two complementary perspectives: (1) a formal methodology that can provide measures of an organization's software acquisition capability, and (2) a set of practices and procedures that can be undertaken to improve the acquisition of software intensive systems. While both perspectives are important to achieve the full benefit from implementation of SA-CMM, we found that most of the resource – intensive requirements stem from it's use as a formal assessment methodology, while most of the benefits accrue from disciplined application it's of practices and procedures

5 Future Directions

The Agency is currently emphasizing performance based contracting and the use of metrics to manage and assess contract performance. A large amount of guidance material is available for formulating software statements of work (SOW) and the selection of software development metrics. Applying this guidance, however, poses a formidable challenge because of the disparate, fragmented, and nonintegrated nature of the information, and also because of the lack of tools to facilitate incorporating this guidance into routine project practice.

The results of this research, along with previous SATC research can be adapted and extended to create an integrated, non-intrusive methodology and toolset to aid projects in software acquisition and performance assessment. Throughout the development process, the toolset could analyze contractor delivered data to identify trends and potential problems in the most common areas of development and product risk.

This new work can also provide the basis for efforts to collect, catalog and maintain data produced by the toolset into a repository. This repository could serve as a resource for project planners, reviewers, and quality assurance staff to anticipate project risks based upon the similarity between a particular project profile and other profiles developed via the tool. In addition, the repository could serve as a valuable learning tool for teams being assembled for new projects and as a novel way for collecting/disseminating lessons learned.

The results of this research will be applicable to all NASA projects that acquire or develop software capabilities. The resources and tool resulting from this activity will provide assistance to project personnel involved in the acquisition of mission software.

Potential users and uses are shown in the table below.

Users	Uses
Project managers	Identification of software acquisition priorities Tracking/monitoring software schedule and resources. Assessment of software progress. .Revision of software priorities
Project Software Manager	Development of software SOW and required deliverable items list. Assessment of software product risks Tracking/monitoring software progress Identification of development trends
Project Performance Assurance Managers	Development of software SOW assurance requirements Tracking/monitoring software quality activities Assessment of software reliability trends
Software COTR	Development of software technical guidance